Chapter 7. Web Services in Action

In the previous chapters, we've been building a picture of the technologies and methodologies around SOAP web services. In this chapter, we apply the discussion to the real-world implementation of a SOAP web service. You'll see how SOAP and WSDL are deployed, and also how to draw in other XML technologies to solve problems that SOAP and WSDL do not address.

The service we'll develop is the CodeShare Service Network, a simple set of peer-to-peer web services for sharing application source code. While we develop that code, we'll stop to take a look at security, and how to implement it when SOAP and WSDL don't cover it.

The CodeShare implementation we show here provides a way for people to share source code. We use digital signatures to verify the identity of clients, and keep a central registry of the files people are offering. Rather than a single web service, the CodeShare application comprises a number of different small interfaces, a common web services design. Each interface can be implemented in any language that supports SOAP, and we used a mixture of Perl and Java to demonstrate this. CodeShare is an example of a peer web service. In the peer-to-peer (P2P) model, the Internet isn't viewed as a network of *clients* accessing the resources of a *server*. Rather, it's a cooperative network of *peers* sharing resources equally and evenly. The lines are blurred between the service provider and the service consumer, with no application required to have just a single role.

Peer web services uses already-deployed web services technologies to provide P2P services.

7.1 The CodeShare Service Network

The CodeShare Service Network is a very simple example of peer web services. It provides an environment where developers may easily share source code with the rest of the world.

7.1.1 Overview

There are three important CodeShare components: the owner of the code being shared, the requester of the code, and the CodeShare server that serves as clearinghouse for the code and as an authentication authority that code owners can use to control access to the code that they are sharing. The relationships between the components are shown in Figure 7-1.

Figure 7-1. The CodeShare architecture

Here is the typical use scenario:

1. The developers of some code decide to share that code publicly. They do so by updating their local project *index.xml* file, indicating the files they wish to share.
2. The developers log onto the CodeShare server to update their entry in the master index maintained at the server.
3. The developers then start their CodeShare owner service (a local SOAP HTTP daemon).
4. Whenever users wish to find code being shared, they have two options: they can connect to the developer's CodeShare owner service directly and execute four basic operations: `search`, `list`, `info`, and `get`; or they can connect to the CodeShare server and search the master index. Doing so will result in a list of all CodeShare owner services sharing code that matches the search request. All `get` operations point directly to the owner service to retrieve the source code being shared.
5. At times, developers may wish to restrict who is allowed to access the code they are sharing. To do so, they simply add the names of all authorized users to their *index.xml* (all users are registered with the CodeShare server). Whenever a user tries to retrieve the restricted code, the owner service will check first to see if the user has logged into the CodeShare server and if so, whether they are allowed access.

7.1.2 Prerequisites

There are a few things that you need to have set up on your system before you can run this example:

SOAP::Lite Version 5.1 and all prerequisites

Instructions on how to install this are given in Chapter 3.

DBI and DBD:CSV

These are Perl SQL database modules used by the CodeShare owner server. Install them by typing `install DBI` and `install DBD::CSV` in the CPAN shell.

A Servlet-enabled web server

We recommend Apache's Jakarta Tomcat Version 3.22. Tomcat can be downloaded from http://jakarta.apache.org/.

Apache Xerces 1.4 or any other JAXP-enabled XML parser

JAXP is the Java API for XML Processing (http://xml.apache.org/xerces-j).

Apache SOAP

At the time of writing, the latest version was 2.2, which has a bug you will need to fix. Download the source distribution of Apache SOAP. The changes and the build process are described in the next section of this chapter.

The latest version of the IBM XML Security Suite

This is available from IBM's alphaWorks web site (http://alphaworks.ibm.com/tech/xmlsecuritysuite). Once downloaded, unzip the distribution and put the *XSS4J.jar* file into your application server's classpath.

7.1.2.1 Fixing the bug in Apache SOAP 2.2

The problem in Version 2.2 of Apache SOAP causes invalid XML to be produced in some situations. CodeShare happens to cause one of those situations. The bug fix detailed here has been submitted and it should not be necessary to make this fix in versions of Apache SOAP after 2.2.

Assuming that you have already downloaded the source distribution of the Apache SOAP source code, locate a file called *DOM2Writer.java*, in the *%SOAP_HOME%\java\src\org\apache\soap\util\xml* folder. (`$SOAP_HOME` will be in the directory where the unzipped contents of the distribution were downloaded.)

At line 172, replace the lines in Example 7-1 with those in Example 7-2.

Example 7-1. Code in DOM2Writer.java to replace

```
out.print(' ' + attr.getNodeName(  ) +"=\"" + normalize(attr.getValue(  ))
+ '\"');
```

Example 7-2. New code for DOM2Writer.java

```
if (attr.getNodeName(  ).startsWith("xmlns:") &&
    !(NS_URI_XMLNS.equals(attr.getNamespaceURI(  )))) {
    String attrName = attr.getNodeName(  );
    String prefix= attrName.substring(attrName.indexOf(":")+1);
    try
        {
            String namespaceURI =
                (String)namespaceStack.lookup(prefix);
            if (!attr.getNodeValue(  ).equals(namespaceURI)) {
                printNamespaceDecl(prefix, namespaceURI,
                                   namespaceStack, out);
            }
        }
    catch (IllegalArgumentException e)
        {
            printNamespaceDecl(prefix,
                               attr.getNodeValue(  ),
                               namespaceStack, out);
        }
    }
else
    {
        out.print('    '    +    attr.getNodeName(        )    +"=\""    +
normalize(attr.getValue(  )) + '\"');
    }
```

Now add the new method in Example 7-3 to the DOM2Writer class.

Example 7-3. New method for the DOM2Writer class

```
private static void printNamespaceDecl(String prefix,
        String namespaceURI, ObjectRegistry namespaceStack,
        PrintWriter out)
{
    if (!(namespaceURI.equals(NS_URI_XMLNS) && prefix.equals("xmlns")))
    {
        out.print(" xmlns:" + prefix + "=\"" + namespaceURI + '\"');
    }
    namespaceStack.register(prefix, namespaceURI);
}
```

Next, compile the Apache SOAP package.

7.1.2.2 Compiling Apache SOAP

To build Apache SOAP, you need to use Ant, a Java build-management tool released by Apache. Ant is available from http://jakarta.apache.org/ and is officially a part of the Jakarta Tomcat project. Once downloaded, please follow the detailed instructions included with the package on how to install it.

Ant uses an XML-based script (*build.xml*) for defining how to compile the code. Apache SOAP's *build.xml* file is located in the *%SOAP_HOME%\java* directory.

Before you can build, you need to make sure that all of the prerequisites are in place. These are listed at the start of the *build.xml* file:

- Any JAXP-enabled XML Parser (Xerces is preferred)
- The JavaMail package, available from http://java.sun.com/products/javamail/
- The Java Activation Framework package, available from http://java.sun.com/products/beans/glasgow/jaf.html

These packages must all be in your classpath prior to attempting the build. Once there, start the build using the following command:

```
java org.apache.tools.ant.Main <target>
```

Where `target` is one of four options:

`compile`

Creates the *soap.jar* package

`javadocs`

Creates the *soap.jar* JavaDocs

`dist`

Creates the complete binary distribution

`srcdist`

Creates the complete source code distribution

For our purposes, use the `compile` target option. This will create a new *soap.jar* file with the modified `DOM2Writer.java` class included. Once built, replace all other *soap.jar* files that may be in your application servers classpath with the newly built *soap.jar*.

7.2 The Code Share Index

The source code shared through the CodeShare network is organized around a simple index structure that preserves the original directory-file hierarchy. Everybody wanting to share source code through the CodeShare must create an index. As an example, let's assume that we are sharing the following Java project:

```
HelloWorld
+---build.xml
+---lib
|   +---HelloWorld.jar
+---src
    +---oreilly
        +---samples
            +---HelloWorld
                +---HelloWorld.java
```

There are a total of six directories and three files being shared. Within the CodeShare index, we represent this project as Example 7-4.

Example 7-4. CodeShare index for sample project

```
<codeShare xmlns:dc="http://purl.org/dc/elements/1.1/">
  <project location="HelloWorld">
    <dc:Title>HelloWorld</dc:Title>
    <dc:Creator>James Snell, et al</dc:Creator>
    <dc:Date>2001-08-20</dc:Date>
    <dc:Subject>Hello World Web service example</dc:Subject>
    <dc:Description>
        Example Hello World Web service
    </dc:Description>
    <file location="build.xml">
      <dc:Title>Ant Build Script</dc:Title>
    </file>
    <directory location="lib">
      <dc:Title>Compiled libraries</dc:Title>
      <file location="HelloWorld.jar">
        <dc:Title>Compiled Hello World JAR</dc:Title>
      </file>
    </directory>
    <directory location="src">
      <dc:Title>Source Code</dc:Title>
      <directory location="oreilly">
        <dc:Title>oreilly</dc:title>
        <directory location="samples">
          <dc:Title>samples</dc:Title>
          <directory location="HelloWorld">
            <dc:Title>HelloWorld</dc:Title>
```

```
            <file location="HelloWorld.java">
              <dc:Title>HelloWorld.java</dc:Title>
            </file>
          </directory>
        </directory>
      </directory>
    </directory>
  </project>
</codeShare>
```

As you can see, the structure of the index is very basic. The `codeShare` element is the root for the entire index. The `project` element defines a shared project. The `directory` element defines a directory being shared within a project. The `file` element defines a file being shared.

The most interesting feature of the index is the use of Dublin Core metadata elements (`dc:Title`, for example) to add descriptive properties to each of the shared items.

The Dublin Core metadata project is an initiative to define standard types of metadata (data about data) capable of describing Internet content. We use it here to provide more flexible searching options when people are looking for particular types of code. Without these descriptive elements, the CodeShare searching capability would be limited to searches based only on the name of the file or directory being searched. Later, we'll see exactly how this additional data is used.

The Dublin Core specification (http://www.dublincore.org/documents/dces/) defines a set of 15 metadata elements, all of which may be used within the CodeShare index. The elements are described in Table 7-1.

Table 7-1. Dublin Core element set

Element name	Element description
`Title`	The name given to the resource
`Creator`	The entity responsible for creating the resource
`Subject`	A short topic that describes the resource
`Description`	A detailed, textual description of the resource
`Publisher`	The entity responsible for making the resource available
`Contributor`	An entity responsible for making contributions to the resource
`Date`	Typically, the date the resource was created
`Type`	The generic type of resource (not the MIME Content Type)
`Format`	The MIME Content Type or other physical format of the resource
`Identifier`	An unambiguous reference to the resource
`Source`	A reference to the resource from which this resource is derived
`Language`	The language (not programming language) in which the resource is presented
`Relation`	A reference to a related resource
`Coverage`	The extent or scope of the resource
`Rights`	Information about rights held in or over the resource

7.3 Web Services Security

What does it mean to add security to web services? In the case of the CodeShare example, our goal is to let the owners of the code specify access rights for particular individuals. If a user is not on the list of approved users, she will not be able to download the code.

Security in web services means adding basic security capabilities to the technologies that make web services happen. This means having the ability to encrypt SOAP messages, digitally sign WSDL service descriptions, add reliability to the protocol transports we use to carry this information around, assert a user's identity, define policies that govern how information is to be used, by whom it can be used, and for what purposes it can be used, and any number of a laundry list of other items. It could take almost an entire book by itself to describe how to implement all of these requirements. Unfortunately, while efforts are currently being made in each of these areas, we are still a long way from having defined standards (de facto or otherwise) on how all of this will happen in the web services environment. For the CodeShare example, we focus on only one: user authentication.

Authentication in SOAP-based web services can occur in a wide variety of ways. The service may choose to use traditional transport-layer authentication methods, such as HTTP Basic or Digest Authentication. Alternatively, the service may choose to implement a service-layer authentication mechanism that makes the service itself responsible for validating a user's identity.

The second approach is what we see emerging in the form of Microsoft's Passport authentication service, which provides Kerberos-based authentication over web service protocols. Kerberos is a popular Internet-standard authentication mechanism based on the exchange of *tickets*. These tickets are used in much the same way as a ticket to a movie. The bearer of the ticket presents it as a pass to get in to see the movie, or in our case, to access a service.

Chapter 8 discusses the Passport authentication scheme and several other alternative approaches in greater detail.

7.3.1 The Security Assertions Markup Language (SAML)

One of the many emerging web service technologies is specifically designed to be used as a method of implementing service-layer global sign-on for web services. The specification, called the Security Assertions Markup Language, or SAML, defines an XML syntax for expressing security-related facts. For example, SAML may be used to express the fact that Pavel Kulchenko authenticated at 10:00 a.m. and that the authentication expires at 2:00 p.m.

SAML assertions, as they are called, are created and digitally signed by the authentication authority who handles the actual authentication process. For example, when a user invokes the login operation on the CodeShare client interface, the CodeShare server (which validates the user ID and password) issues the SAML assertion stating that the login was successful. By digitally signing that assertion, anybody who receives it may validate that it was, in fact, created and issued by the CodeShare server.

Example 7-5 is a digitally signed SAML assertion returned by the login operation. The assertion itself is highlighted in bold type. The first part of this structure is the XML Digital

Signature, which validates that the SAML assertion is authentic. XML Digital Signatures are being standardized through a joint effort by the W3C and the IETF. The structure of these signatures is too complex to explain here, so we've provided links to some supplemental information in Chapter 8. Luckily, we do not have to create these signatures manually. This particular example was created using IBM's XML Security suite.

Example 7-5. SAML assertion

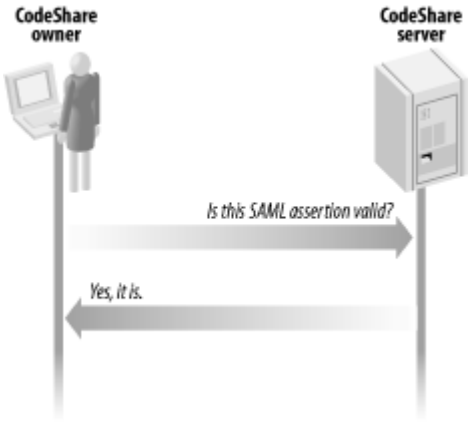
```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
           Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-20000119"/>
    <SignatureMethod
           Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="#999852828470">
      <DigestMethod
           Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>pCvvhLY/UdR7D8Jzja7kG2+finQ=</DigestValue>
    </Reference>
  </SignedInfo>
 <SignatureValue>
    T110Nd9tt4f1m9Ahoe82HoPXWrZ0se/9ON9qU01TRkZ4FrOg8DBg9g==
 </SignatureValue>
 <KeyInfo>
   <KeyValue>
     <DSAKeyValue>
       <P>
        /X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow9s
        ubVWzXgTuAHTRv8mZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bT
        xR7DAjVUE1oWkTL2dfOuK2HXKu/yIgMZndFIAcc=         </P>
       <Q>l2BQjxUjC8yykrmCouuEC/BYHPU=</Q>
       <G>
        9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFn
        Ej6EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTx
        vqhRkImog9/hWuWfBpKLZl6Ae1UlZAFMO/7PSSo=         </G>
       <Y>
        xbzyPw8CzjbnzxmoB9WDLnR0Enw2/5CxHLsozIXNT+n/EtZpi3okfytFxjAcQVUuiZ
        Jwkf2/Eke7peA/R5dd9krb1j0EdlTVXd+eOcyWJOWplKEJuNYclrC4f+zy6FTcxGlq
        d/GqVEwud1kUiQ+5RPoAYsxpzaRDAVIeaarxXN0=         </Y>
     </DSAKeyValue>
   </KeyValue>
   <X509Data>
     <X509IssuerSerial>
       <X509IssuerName>CN=Codeshare</X509IssuerName>
       <X509SerialNumber>999849441</X509SerialNumber>
     </X509IssuerSerial>
     <X509SubjectName>CN=Codeshare</X509SubjectName>
     <X509Certificate>

MIICXjCCAhsCBDuYfeEwCwYHKoZIzjgEAwUAMBQxEjAQBgNVBAMTCUNvZGVzaGFyZTAeFw0wMTA
5MDcwNzU3MjFaF
w0wMTEyMDYwNzU3MjFaMBQxEjAQBgNVBAMTCUNvZGVzaGFyZTCCAbgwggEsBgcqhkjOOAQBMIIB
HwKBgQD9f1OBHX
USKVLfSpwu7OTn9hG3UjzvRADDHj+AtlEmaUVdQCJR+1k9jVj6v8X1ujD2y5tVbNeBO4AdNG/
yZmC3a5lQpaSfn+gEexAiwk+7qdf+t8Yb+DtX58aophUPBPuD9tPFHsMCNVQTWhaRMvZ1864rYd
cq7/
IiAxmd0UgBxwIVAJdgUI8VIwvMspK5gqLrhAvwWBz1AoGBAPfhoIXWmz3ey7yrXDa4V7l5lK+7+
jrqgvlXTAs9B4J
nUVlXjrrUWU/
```

```
mcQcQgYC0SRZxI+hMKBYTt88JMozIpuE8FnqLVHyNKOCjrh4rs6Z1kW6jfwv6ITVi8ftiegEkO8
yk8b6oUZCJqIPf
4VrlnwaSi2ZegHtVJWQBTDv+z0kqA4GFAAKBgQDFvPI/DwLONufPGagH1YMudHQSfDb/
kLEcuyjMhc1P6f8S1mmLeiR/K0XGMBxBVS6JknCR/
b8SR7ul4D9Hl132StvWPQR2VNVd3545zJYk5amUoQm41hyWsLh/
7PLoVNzEaWp38apUTC53WRSJD7lE+gBizGnNpEMBUh5pqvFc3TALBgcqhkjOOAQDBQADMAAwLQI
VAIyej/
xrPI4jpVCBUdHz/zz4nUY9AhRGb/VRBiqS2NKo+PO0KbURVg2g5A==   </X509Certificate>
    </X509Data>
   </KeyInfo>
   <dsig:Object Id="999852828470" xmlns=""
                xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     <AuthenticationAssertion AssertionID="999852828470"
                  IssueInstant="Fri Sep 07 01:53:48 PDT 2001"
                  Issuer="CodeShare.org" Version="1.0"
                  xmlns="http://www.oasis-
open.org/committees/security/docs/
                        draft-sstc-schema-assertion-15.xsd">

       <Subject>
         <NameIdentifier>
           <SecurityDomain>CodeShare.org</SecurityDomain>
           <Name>james</Name>
         </NameIdentifier>
       </Subject>
       <AuthenticationMethod>http://codeshare.org</AuthenticationMethod>
       <AuthenticationInstant>
           Fri Sep 07 01:53:48 PDT 2001
       </AuthenticationInstant>
       <AuthenticationLocale>
         <IP>123.123.123.123</IP>
         <DNS_Domain>codeshare.org</DNS_Domain>
       </AuthenticationLocale>
     </AuthenticationAssertion>
  </dsig:Object>
</Signature>
```

The purpose of this example SAML assertion is to state that the user `james` from the domain `CodeShare.org` authenticated on Friday, September 7, at 1:53 p.m. Pacific Daylight Time 2001, using CodeShare's default authentication method (the login operation). The authentication itself was provided by a server located at the `123.123.123.123` IP address with the DNS domain name `codeshare.org`. This statement is digitally signed using the CodeShare Servers X509 digital certificate, guaranteeing its authenticity.

When a user presents this token to a CodeShare owner, the owner can verify that it is authentic by asking the CodeShare server if it really did issue the statement. Figure 7-2 illustrates the flow of messages.

Figure 7-2. A flow illustrating the typical conversation between the CodeShare owner and CodeShare server



SAML assertions can be created and validated by anybody, making them a very good mechanism for implementing single sign-on functionality. Later in this chapter, we will demonstrate how this SAML assertion was created and signed.

7.4 Definitions and Descriptions

Because web services are all about the interfaces that tie applications together, the first thing we need to do is define what those interfaces look like.

In this example there are four interfaces of interest:

- The owner interface implemented by CodeShare owners and the CodeShare server for allowing users to search for and retrieve source code.
- The client interface implemented by the CodeShare server for allowing users to register and login.
- The login verification interface implemented by the CodeShare server to allow CodeShare owners to ensure that users have logged on.
- The master index interface that allows CodeShare owners to update their entries in the master index maintained by the CodeShare server.

Each of these interfaces is expressed using WSDL service interface descriptions.

7.4.1 The Owner Interface

The owner interface consists of four fundamental operations:

`search`

Searches the *index.xml* file for elements that match a given value. By default, this operation searches only the Dublin Core `Title` element, but other Dublin Core elements may be targeted instead.

If this was a Java function, it would be something like:

```
public List search(String value,
                   String dcElement);
```

In this example, `dcElement` equals the name of the Dublin Core element that you want to search.

`list`

Lists all of the projects and items being shared by the owner. Only basic information about the items is returned, including the location and title of the items. Filters may be applied that return only items that match a given value of a specified Dublin Core element. Like the `search` operation, the title is the default element upon which filters are applied.

Again looking at this from a Java perspective, the signature of this operation is:

```
public List list(String value,
                 String dcElement);
```

Unlike the `search` operation, however, both the `value` and `dcElement` parameters are optional.

`info`

Returns detailed information about each of the projects and items being shared. Like the `list` operation, filters may be applied to limit the number of items that are returned. The signature of this operation is exactly the same as the `list` operation.

`get`

Returns all of the files being shared for a specified project or projects. The exact directory structures will be recreated. The signature of this operation is also the same as the `list` operation.

All of these operations return a SOAP encoded array of `item` elements similar to Example 7-6.

Example 7-6. Sample array returned by owner operations

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
              xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/">
  <env:Body>
    <env:listResponse>
      <Items enc:arrayType="csi:item[2]"
             xsi:type="csi:ArrayOfItems">
        <item xsi:type="namesp1:SOAPStruct">
          <path xsi:type="xsd:string">HelloWorld</path>
          <title xsi:type="xsd:string"> HelloWorld </title>
          <fullpath xsi:type="xsd:string">HelloWorld/</fullpath>
          <type xsi:type="xsd:string">project</type>
        </item>
        <item xsi:type="namesp1:SOAPStruct">
          <path xsi:type="xsd:string" />
```

```
          <title xsi:type="xsd:string">build.xml</title>
          <fullpath xsi:type="xsd:string">HelloWorld/</fullpath>
          <type xsi:type="xsd:string">file</type>
        </item>
      </enc:Array>
    </env:listResponse>
  </env:Body>
</env:Envelope>
```

7.4.1.1 WSDL port type definition

The full WSDL port type definition is given in Appendix C. We'll cover the highlights here: the data types, the messages, the port type, and the protocol binding.

7.4.1.2 Data types

The data types are defined with an embedded XML schema. This schema defines two data types: an `item`, which, as we saw in the previous SOAP envelope, represents a project item within the CodeShare index, and an array of items.

The `item` definition, shown in Example 7-7, is straightforward. It is a complex type element with four children and a flag to include any Dublin Core elements that the CodeShare service may want to add.

Example 7-7. The item definition

```
<xsd:element name="item">
  <xsd:annotation>
    <xsd:documentation>
      CodeShare Indexed Item
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:all>
        <xsd:element name="path" type="xsd:string"
                     nullable="true" minOccurs="0"/>
        <xsd:element name="title" type="xsd:string"
                     nullable="true" minOccurs="0"/>
        <xsd:element name="fullpath" type="xsd:string"
                     nullable="true" minOccurs="0"/>
        <xsd:element name="type" type="xsd:string"
                     nullable="true" minOccurs="0"/>
      </xsd:all>
      <xsd:any namespace='xmlns:dc="http://purl.org/dc/elements/1.1/"'
               processContents="lax" minOccurs="0"
               maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

The `ArrayOfItems` data type, given in Example 7-8, is a derivative of the `Array` data type defined by the SOAP Section 5 encoding style. With this definition, we state this is an array of `item` elements as specified by the Section 5 encoding rules.

Example 7-8. The ArrayOfItems definition

```
<xsd:complexType name="ArrayOfItems">
  <xsd:annotation>
    <xsd:documentation>
      Array of CodeShare item elements
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="se:Array">
      <xsd:attribute ref="se:arrayType"
                     wsdl:arrayType="types:item[]" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

7.4.1.3 Messages

There are exactly two messages defined for each operation. A sample operation's messages are defined in Example 7-9. See the complete listing in Appendix C.

Example 7-9. Search message definitions

```
<wsdl:message name="search">
    <part name="p1" type="xsd:string" />
    <part name="p2" type="xsd:string" />
</wsdl:message>
<wsdl:message name="searchResponse">
    <part name="response" type="types:ArrayOfItems" />
</wsdl:message>
```

7.4.1.4 Port type

The owner interface port type is defined in terms of the messages we just created. Example 7-10 shows the `portType` element with a representative operation defined. See Appendix C for the complete listing. The `parameterOrder` attribute is a WSDL mechanism for specifying the order in which the parts of a message must appear within the body of the SOAP message.

Example 7-10. The portType definition

```
<wsdl:portType name="CodeShareOwnerInterface">
    <wsdl:operation name="search" parameterOrder="p1 p2">
        <wsdl:input name="search" message="tns:search" />
        <wsdl:output name="searchResponse"
                     message="tns:searchResponse" />
    </wsdl:operation>
    <!-- and so on for the other operations -->
</wsdl:portType>
```

7.4.1.5 Protocol binding

Example 7-11 specifies that the owner interface port type is accessed via SOAP messages transported over HTTP. Each of the SOAP messages will conform to the Section 5 encoding style (as indicated by the `soap:body` elements). As before, only one operation is shown here. For the full set, see the complete WSDL listing in Appendix C.

Example 7-11. Binding the interface to the portType

```
<wsdl:binding name="CodeShareOwner_SOAP_HTTP"
              type="tns:CodeShareOwnerInterface">
    <soap:binding style="rpc"
     transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="search">
        <soap:operation soapAction="urn:CodeShareOwner#search" />
        <wsdl:input>
            <soap:body use="encoded" namespace="urn:CodeShareOwner"
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </wsdl:input>
        <wsdl:output name="Name">
            <soap:body use="encoded" namespace="urn:CodeShareOwner"
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </wsdl:output>
    </wsdl:operation>
    <!-- and so on for the other operations -->
/wsdl:binding>
```

7.4.2 The Client Interface

The client interface consists of two operations: `register` and `login`. Because the WSDL interface is similar in structure to the one defined for the owner interface, we will not show it here. It is printed in full in Appendix C.

`register`

> Takes a simple user ID and password pair to create a new user account on the CodeShare server.

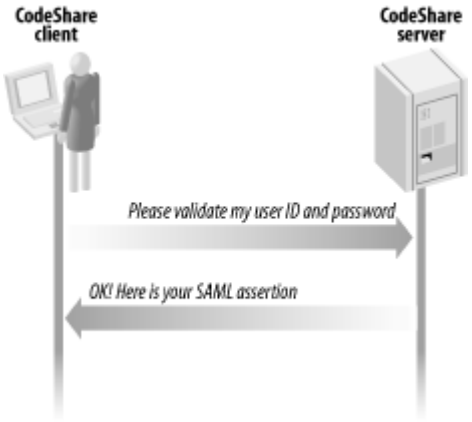
`login`

> Receives a Base64 encoded string consisting of the user ID and password and returns a digitally signed SAML assertion indicating that the user logged in successfully. If login was not successful, a SOAP fault with the type "Client.Authentication" will be returned.

7.4.2.1 CodeShare login operation

The CodeShare login operation validates the user ID and password and generates a signed SAML assertion, as shown in Figure 7-3. It's not a perfect security solution—the SAML specification is missing some very significant pieces (for example, it is very easy for somebody to intercept a signed SAML assertion and pretend to be the person for whom it is issued). It will be some time before all of these issues get worked out. For our purposes, we need only something simple, just to demonstrate the basic idea.

Figure 7-3. A flow illustrating the typical conversation between the CodeShare client and CodeShare server



7.4.3 The Login Verification Interface

The user presents the CodeShare server's SAML assertions to retrieve the code being shared. The assertion is given to the code owner who must validate that it did in fact come from the CodeShare service. The CodeShare login verification interface provides this functionality.

There is only a single `verify` operation defined by this interface. It takes the SAML assertion as an input and returns a simple true or false value indicating the validity of the assertion. See Appendix C for the full WSDL.

7.4.4 The Master Index Interface

Aside from providing the user management and authentication functions for the CodeShare network, the CodeShare service also provides a master index of all code being shared. Code owners update this master index through the master index interface.

This interface defines two operations. New code owners participate in the CodeShare network through `register`, and the `update` operation lets owners update their entries in the master index. The update operation receives the owner's user ID, password, and up-to-date project index.

7.5 Implementing the CodeShare Server

The CodeShare server is implemented as a set of Java classes that maintain a master index of every CodeShare owner who is sharing code and all of the registered users who may have access to that code. The server is divided into four distinct web services, each an implementation of the four interfaces that we've already defined: the master index service, the owner service, the client service, and the verification service.

7.5.1 The Master Index Service

The master index service allows CodeShare owners to update their entries in the index maintained by the Code Share server. The `codeshare.OwnerService` class implements the index service.

7.5.1.1 Operations

The list of registered owners is stored as an XML file. The `register` operation in Example 7-12 simply adds a new element to that XML file.

Example 7-12. The register operation

```
public static boolean register(String ownerid, String password, String url)
{
    Element e = doc.getDocumentElement(  );
    NodeList nl = e.getElementsByTagName("owner");
    for (int n = 0; n < nl.getLength(  ); n++) {
        Element ex = (Element)nl.item(n);
        if (ex.getAttribute("id").equals(ownerid)) {
            throw new IllegalArgumentException(
                "An owner with that ID already exists!");
        }
    }
    Element u = doc.createElement("owner");
    u.setAttribute("id", ownerid);
    u.setAttribute("password", password);
    u.setAttribute("url", url);
    e.appendChild(u);
    XMLUtil.put(owners, doc);
    return true;
}
```

The master index itself (the list of all code being shared through the CodeShare network) is also maintained as an XML file. As with the `register` operation, the `update` operation shown in Example 7-13 does nothing more than update this XML file by either inserting the index passed in by the owner, or replacing an existing part of the index updated at a previous time.

Example 7-13. The update operation

```
public static boolean update(String ownerid, String password, Element
index) {
    Element el = doc.getDocumentElement(  );
    NodeList nl = el.getElementsByTagName("owner");
    for (int n = 0; n < nl.getLength(  ); n++) {
        Element e = (Element)nl.item(n);
        if (e.getAttribute("id").equals(ownerid) &&
            e.getAttribute("password").equals(password)) {
            Element i = (Element)doc.importNode(index, true);
            NodeList c = e.getElementsByTagName("index");
            if (c.getLength(  ) > 0) {
                Node node = c.item(1);
                e.replaceChild(node, i);
            } else {
                e.appendChild(i);
            }
            XMLUtil.put(owners, doc);
            return true;
        }
    }
    return false;
}
```

7.5.1.2 Deployment

This service is deployed to the Apache SOAP engine using the process we described in Chapter 3. The deployment descriptor we'll use is shown in Example 7-14.

Example 7-14. The deployment descriptor for the master index service

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
             id="urn:CodeShareService-MasterIndex">
  <isd:provider type="java"
                scope="Application"
                methods="register update">
    <isd:java class="codeshare.IndexService"/>
  </isd:provider>
  <isd:faultListener>org.apache.soap.server.DOMFaultListener
  </isd:faultListener>
</isd:service>
```

7.5.2 The Owner Service

The owner service is a partial implementation of the CodeShare owner interface. This interface allows users to search the master index, but does not allow them to use the `get` or `info` operations—these must be performed directly against the CodeShare owner service that is providing the code. The CodeShare server will provide the information users need to find the owner services sharing the code they wish to access.

The operations implemented by the owner service (`search` and `list`) basically do the same thing: return an array of shared items that match the specified criteria. The search operation is shown in Example 7-15. This operation loops through all of the items in the master index looking for matching items.

Example 7-15. The owner service operations

```
public org.w3c.dom.Element search(String p1)
{
    return search(p1, "dc:Title");
}

public Element search(String p1, String p2)
{
  Element e = doc.getDocumentElement(  );
  NodeList nl = e.getElementsByTagName(p2);

  Document d = SAMLUtil.newDocument(  );
  Element list = doc.createElement("list");
  d.appendChild(list);

  for (int n = 0; n < nl.getLength(  ); n++)
  {
    Element next = (Element)nl.item(n);
```

```
    try
      {
        RE targetRE = new RE(p1);
        if (targetRE.match(SAMLUtil.getInnerText(next.getText(  )))
        {
          Element item = (Element)d.importNode(next);
          list.appendChild(item);
        }
      }
    catch (Exception exc)
      {
      }
  }
  return list;
}
```

The CodeShare server does not support the `info` and `get` operations, so we do not need to write any code to implement them at this point.

This service is deployed to the Apache SOAP engine using the same process we described in Chapter 3. The deployment descriptor we'll use is shown in Example 7-16.

Example 7-16. Deployment descriptor for the CodeShare server

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
             id="urn:CodeShareService-OwnerService">
  <isd:provider type="java"
                scope="Application"
                methods="list search">
    <isd:java class="codeshare.OwnerService"/>
  </isd:provider>
  <isd:faultListener>org.apache.soap.server.DOMFaultListener
  </isd:faultListener>
</isd:service>
```

7.5.3 The Client Service

The client service is a Java implementation of the client interface that allows users to register and log in to the CodeShare service. This service keeps track of all the various user accounts and issues the digitally signed SAML assertions that users present to the CodeShare owners. Owners use those assertions to ensure that only authorized users can access the code they are sharing.

7.5.3.1 Creating SAML assertions

Creating a SAML assertion involves nothing more than creating an XML document that conforms to the SAML schema. Because SAML is still being developed, this application implements just a small subset of the most recent working draft being discussed within the SAML working group.

We create the SAML assertion by building an object model (which we created; there currently is no standard SAML API) that corresponds to each of the major parts of the SAML schema, then serializing that object model out to an XML document. The full code for this example is available in Appendix C. Here we just show the assertion being created.

We use the `AssertionFactory` class to create an instance of the object model. This class, given in full in Appendix C, is discussed next.

The first step is to set the various required properties, such as the assertion ID, the name of the issuer, and the data and time at which the assertion was created. Example 7-17 shows this step.

Example 7-17. Setting assertion properties

```
AuthenticationAssertion aa =
  new AuthenticationAssertion(  );

IDType aid = new IDType(id);
aa.setAssertionID(aid);
aa.setIssuer(issuerName);
aa.setIssueInstant(issueInstant);
```

Every SAML assertion has a subject that indicates what the assertion is about. In this case, the subject is a name identifier that states which user has been authenticated. Example 7-18 shows how to set the subject.

Example 7-18. Setting the SAML subject

```
Subject subject = new Subject(  );
{
    NameIdentifier ni = new NameIdentifier(  );
    ni.setName(name);
    ni.setSecurityDomain(domain);
    subject.setNameIdentifier(ni);
    aa.setSubject(subject);
}
```

Finally, we fill in additional details about the type of authentication that was used, and an indication of where the authentication occurred (Example 7-19).

Example 7-19. Completing the authentication assertion

```
aa.setAuthenticationMethod(
new AuthenticationMethod(method));
aa.setAuthenticationInstant(new AuthenticationInstant(authInstant));
AuthenticationLocale locale = new AuthenticationLocale(  );
locale.setIP(ip);
locale.setDNSDomain(dns);
aa.setAuthenticationLocale(locale);
```

The `AssertionFactory` provides a convenient wrapper to this dogwork. To create an assertion, simply pass in the various relevant pieces of information, and the authentication assertion is created, filled out, and returned.

Example 7-20 uses the `AssertionFactory` class to generate a SAML assertion.

Example 7-20. Using the AssertionFactory class

```
AuthenticationAssertion aa = AssertionFactory.newInstance(
    new String( new Long(System.currentTimeMillis()).toString() ),
    "CodeShare.org",
    new java.util.Date(  ),
    userid,
    "CodeShare.org",
    "http://codeshare.org",
    java.net.InetAddress.getLocalHost().getHostAddress(),
    java.net.InetAddress.getLocalHost().getHostName()
  );
```

Example 7-21 shows the SAML assertion created by Example 7-20.

Example 7-21. The SAML assertion

```
<AuthenticationAssertion
      AssertionID="999852828470"
      IssueInstant="Fri Sep 07 01:53:48 PDT 2001"
      Issuer="CodeShare.org" Version="1.0"
      xmlns="http://www.oasis-open.org/committees/security/docs/draft-
            sstc-schema-assertion-15.xsd">
 <Subject>
   <NameIdentifier>
     <SecurityDomain>CodeShare.org</SecurityDomain>
     <Name>james</Name>
   </NameIdentifier>
 </Subject>
 <AuthenticationMethod>
    http://codeshare.org
 </AuthenticationMethod>
 <AuthenticationInstant>
    Fri Sep 07 01:53:48 PDT 2001
 </AuthenticationInstant>
 <AuthenticationLocale>
   <IP>127.0.0.1</IP>
   <DNS_Domain>diamond</DNS_Domain>
 </AuthenticationLocale>
</AuthenticationAssertion>
```

This is the assertion in Example 7-5.

7.5.3.2 Java keystores

Signing the assertion will use Java's support for public and private keys, which warrants a quick refresher. Somewhere on your computer is a Java keystore, a local database where all of your private keys are stored. You create a new key in this database by using the `keytool` utility that ships with Java. You can also use `keytool` to create new keystore databases.

This command creates the private key for the CodeShare server, which is used to sign all SAML assertions.

```
C:\book>keytool -genkey -dname "cn=CodeShare Server" -keypass CodeShare -
alias
CodeShare -storepass CodeShare -keystore codeshare.db
```

This creates a new file called *codeshare.db* in the *C:\book* folder that contains the private key corresponding to the `cn=CodeShare Server` distinguished name.

7.5.3.3 Signing the SAML assertion

We use the IBM XML Security Suite's XML Digital Signature capability to sign the SAML assertion. The full source to the `AssertionSigner` class is given in Appendix C. The highlights are discussed in this section. At the time of writing, the programming interface of the IBM XML Security Suite was being redesigned. The code shown here was tested with Version XYZ of the XML Security Suite but may not work with other versions.

We can't sign an `assertion` object directly. Instead we have to serialize it to a DOM document and sign that. The serialization is handled by the code in Example 7-22.

Example 7-22. Serializing the assertion to a DOM document

```
Document doc = SAMLUtil.newDocument(  );
Element root = doc.createElement("root");
assertion.serialize(root);
```

A `SignatureGenerator` object will make the signature for us. In the constructor, we indicate the encryption and signing protocols we want to use. This is shown in Example 7-23.

Example 7-23. Creating a SignatureGenerator

```
SignatureGenerator siggen =
  new SignatureGenerator(doc,
    DigestMethod.SHA1,
    Canonicalizer.W3C,
    SignatureMethod.DSA,
    null);
```

The XML signature document can include the message being signed, or the message might be linked as an external resource or another XML document. The most common approach is to include the message being signed—so that's what we'll do, as shown in Example 7-24.

Example 7-24. Embedding the message being signed

```
siggen.addReference(
  siggen.createReference(
    siggen.wrapWithObject(
      root.getFirstChild(  ),
      assertion.getAssertionID(  ).getText(  ))
    )
  );
```

The code in Example 7-25 accesses the keystore and extracts the information it needs to prepare the key needed for signing the SAML assertion. Also prepared is an X509 digital certificate that includes the public key for the `cn=CodeShare Server`. This certificate is embedded into the signature so that it can be used later to validate the signature.

Example 7-25. Preparing the key

```
KeyStore keystore = KeyStore.getInstance("JKS");
keystore.load(
  new FileInputStream(keystorepath),
  storepass.toCharArray(  ));
X509Certificate cert = (X509Certificate)keystore.getCertificate(alias);
Key key = keystore.getKey(alias, keypass.toCharArray(  ));
if (key == null) {
  throw
  new IllegalArgumentException("Invalid Key Info");
}
KeyInfo keyInfo = new KeyInfo(  );
KeyInfo.X509Data x5data = new KeyInfo.X509Data(  );
x5data.setCertificate(cert);
x5data.setParameters(cert, true, true, true);
keyInfo.setX509Data(new KeyInfo.X509Data[] { x5data });
keyInfo.setKeyValue(cert.getPublicKey(  ));
siggen.setKeyInfoGenerator(keyInfo);
```

Finally, we can use the prepared key to sign the assertion. The `sign` operation on the `SignatureContext` object handles the complex cryptographic processes to create the signature (see Example 7-26). Once the assertion has been signed, we return the DOM element that contains the signature just created.

Example 7-26. Signing the assertion

```
Element sig = siggen.getSignatureElement(  );
SignatureContext context = new SignatureContext(  );
context.sign(sig, key);
return sig;
```

7.5.3.4 The login operation

The encryption code is used in the `login` operation, shown in Example 7-27. It verifies the user's password, generates the SAML assertion, and signs it.

Example 7-27. The login operation

```
public static Element login(String userid,
                            String password) throws Exception {

    Element el = doc.getDocumentElement(  );
    NodeList nl = el.getElementsByTagName("user");
    for (int n = 0; n < nl.getLength(  ); n++) {
        Element e = (Element)nl.item(n);
        if (e.getAttribute("id").equals(userid) &&
            e.getAttribute("password").equals(password)) {

            AuthenticationAssertion aa =
                AssertionFactory.newInstance(new String(new
                      Long(System.currentTimeMillis()).toString(  )),
                      "CodeShare.org",
                      new java.util.Date(  ),
                      userid,
                      "CodeShare.org",
                      "http://codeshare.org",
                      new java.util.Date(  ),
                      java.net.InetAddress.getLocalHost(  ).
                          getHostAddress(  ),
```

```
                    java.net.InetAddress.getLocalHost().getHostName(  ));

            Element sa = AssertionSigner.sign(aa,
                                "CodeShare.db",
                                "CodeShare",
                                "CodeShareKeyPass",
                                "CodeShareStorePass");
            return sa;
        }
    }
    return null;
}
```

The deployment descriptor we use to deploy the client service to Apache Axis is shown in Example 7-28.

Example 7-28. Deployment descriptor

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
             id="urn:CodeShareService-ClientService">
  <isd:provider type="java"
                scope="Application"
                methods="register login">
    <isd:java class="codeshare.AuthenticationService"/>
  </isd:provider>
  <isd:faultListener>org.apache.soap.server.DOMFaultListener
  </isd:faultListener>
</isd:service>
```

7.5.4 The Verification Service

The task of the verification service is to validate the signed SAML assertion on behalf of the CodeShare owner. The CodeShare server handles this validation for the owner, so the owner doesn't have to worry about the complexities of implementing the full XML digital signature specification. Granted, it's not a very secure approach, but it works for our purposes here.

The verification service exposes a single operation, `verify`, which receives the signature and returns a Boolean response indicating whether that signature is valid (see Example 7-29). A real verification of the SAML assertion would include a number of checks, such as ensuring that all of the fields contain valid data. We have omitted those checks in the interest of brevity.

Example 7-29. The verify operation

```
public static boolean verify(Element signature) throws Exception {

    Key key = null;
    Element keyInfoElement = KeyInfo.searchForKeyInfo(signature);
    if (keyInfoElement != null) {
        KeyInfo keyInfo = new KeyInfo(keyInfoElement);
        key = keyInfo.getKeyValue(  );
    }
    SignatureContext context = new SignatureContext(  );
    Validity validity = context.verify(signature, key);
    return validity.getCoreValidity(  );
}
```

Deploy this service using the deployment descriptor in Example 7-30.

Example 7-30. Deployment descriptor for the verification service

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
             id="urn:CodeShareService-Verification">
  <isd:provider type="java"
                scope="Application"
                methods="verify">
    <isd:java class="codeshare.VerificationService"/>
  </isd:provider>
  <isd:faultListener>org.apache.soap.server.DOMFaultListener
  </isd:faultListener>
</isd:service>
```

7.6 Implementing the CodeShare Owner

The CodeShare owner server is a lightweight Perl application that consists of two parts: the owner module and a SOAP-enabled HTTP daemon. The server builds on top of SOAP::Lite.

7.6.1 The Owner Module

The CodeShare::Owner module works with the owner's *index.xml* to allow users to search for and retrieve shared code. The owner service also interacts with the CodeShare service to update the master index and validate the identities of users who submit SAML assertions.

This code, written in Perl, implements the same owner interface as the Java CodeShare owner service seen in the previous example. The same WSDL interface description applies to both. The operations have the same effect and return the same types of data.

The `init` method shown in Example 7-31 turns the owner's *index.xml* into a data structure, stored in the variable `$index`.

Example 7-31. The init method

```
sub init {
  my($class, $root) = @_;
  open(F, $root) or die "$root: $!\n";
  $index = SOAP::Custom::XML::Deserializer
           ->deserialize(join '', <F>)->root;
  close(F) or die "$root: $!\n";
}
```

To interact with the CodeShare server master index and validation services, we create a SOAP::Lite proxy to the CodeShare server (shown in Example 7-32).

Example 7-32. Constructing a SOAP::Lite proxy to the server

```
my $codeshare_server;
sub codeshare_server {
  return $codeshare_server ||=
         SOAP::Lite
           ->proxy($SERVER_ENDPOINT)
           ->uri("urn:Services:CodeShareServer");
}
```

The `update` operation shown in Example 7-33 updates this owner's entry in the master index with the information given as arguments to the method.

Example 7-33. The update operation

```
sub update {
  shift->codeshare_server->update(@_)->result;
}
```

When a user submits a SAML assertion, it must be validated by the owner service. The code in Example 7-34 uses the CodeShare service proxy created in Example 7-32 to do just that. Once the assertion is validated, it is cached so the owner doesn't have to validate it again.

Example 7-34. Validating an assertion

```
sub is_valid_signature {
  my($self, $username, $signature) = @_;

  my $key = join "\0", $username, $signature;

  # already cached?
  return $cache{$key} if exists $cache{$key};

  my $response = eval { $self->codeshare_server
     ->isValid(SOAP::Data->type(xml => $signature)) };
  die "CodeShare server is unavailable. Can't validate credentials\n" if
$@;
  die "CodeShare server is unavailable. ",
      $response->faultstring, "\n" if $response->fault;
  die "Invalid credentials\n"
      unless $cache{$key} = $response->result;

  return $cache{$key};
}
```

The `traverse` procedure shown in Example 7-35 navigates the *index.xml*, checking whether the items in the index match the criteria requested by the user in the `search`, `info`, `list`, or `get` operations.

Example 7-35. The traverse method

```
sub traverse {
  my($self, %params) = @_;

  my $start = $params{start};

  my $type = $start->SOAP::Data::name; # file|project|directory
  my $location = ref $start->location ? $start->location->value : '';

  # path to current structure. Empty for projects
  my $path = $type eq 'directory' ||
             $type eq 'file' ? join('/', $params{path} || (  ), $location)
              : '';
  my $prefix = $type eq 'project' ? $location : $params{prefix} || '';
  my $fullpath = join '/', $prefix, $path; # full path. Used to GET files

  my $where = $params{where};
```

```
  my $matched =
    $params{get} && $params{matched} ||
    $params{what} &&
    # check only subelements in Dublin Core namespace
    $start->$where()  =~  /$params{what}/  &&  $start->$where(    )->uri  eq
$DC_NS;

  return
    # current element
    ($matched
       ? +{ type => $type,
            path => $path,
            ($params{get} ? (fullpath => $fullpath) : (  )),
            map { ref $start->$_() ? ($_ => $start->$_()->value) : (  )
                } @ELEMENTS
          }
       : (  )
    ),

    # and everything below
    map { $self->traverse(start => $_, where => $where,
          what    => $params{what},
          path => $path,
          prefix  => $prefix,
      get => ($params{get} || 0),
          matched => $matched) }
         $start->project, $start->directory,
         ($type eq 'file' ? (  ) : $start->file)
  ;
}
```

The `list` operation provides a simple listing of all shared items, and is shown in Example 7-36.

Example 7-36. The list method

```
sub list {
  pop;
  my($self, $what) = @_;
  return [ map { my $e = $_; +{ map {$_ => $e->{$_}}
                                qw(type path Title file fullpath) } }
      $self->traverse(start => $index, where => 'Title',
                      what  => $what,  get   => 1)
  ];
}
```

The `get` operation shown in Example 7-38 retrieves the requested set of items; before it can do so, however, it must check to see if the user is authorized to access those items. It does so by validating the SAML assertion of the user and checking to see if the owner has explicitly allowed the user to access.

The owner specifies access permissions for a specific item in the index by adding a Dublin Core `dc:Rights` element to it. The value of this element is the list of users allowed to access it. If the element is missing, it is assumed that everyone is allowed to access it. Example 7-37 shows a sample index file.

Example 7-37. Sample index file

```
<codeshare>
  <project location="HelloWorld">
    <dc:Title>Hello World</dc:Title>
    <dc:Rights>james pavel doug</dc:Rights>
  </project>
</codeshare>
```

The index in Example 7-37 indicates that the users `james`, `pavel`, and `doug` are allowed to get the HelloWorld project item. The `get` operation shown in Example 7-38 retrieves the requested set of items.

Example 7-38. The get operation

```
sub get {
  my $self = shift;
  my $envelope = $_[-1];
  my $username =
  $envelope->valueof('//{http://www.oasis-open.org/committees/security/
docs/draft-sstc-schema-assertion-15.xsd}Name');
  my $results = $self->list(@_);
  [ map { # return file
      $_->{type} eq 'file' && open(F, delete $_->{fullpath})
        ? ($_->{file} = join('', <F>), close F) : (  ); $_
    }
    grep { # check rights
      ($_->{Rights} || '') =~ /^\s*$/ || # public access if empty
       $username && $_->{Rights} =~ /\b$username\b/ &&
         $self->is_valid_signature($username, get_signature($envelope))
    }
      @$results
  ];
}
```

7.6.2 The Server Daemon

To deploy this code as a web service, simply create and run the HTTP server daemon given in Example 7-39.

Example 7-39. The HTTP server daemon

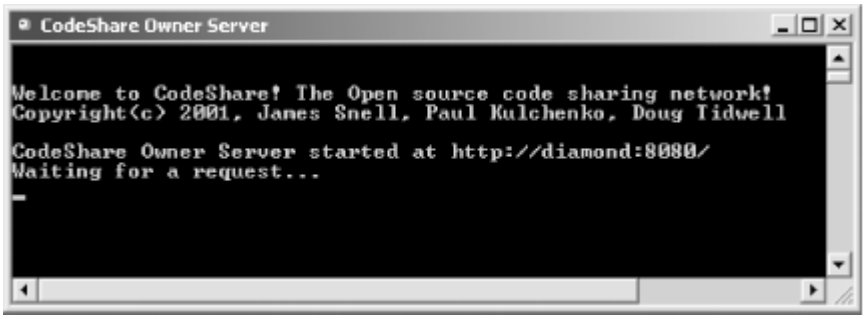
```
use SOAP::Transport::HTTP;
use CodeShare::Owner;

print "\n\nWelcome to CodeShare! The Open source code sharing network!";
print "\nCopyright(c) 2001, James Snell, Pavel Kulchenko, Doug Tidwell\n";

CodeShare::Owner->init(shift or die "Usage: $0 <path/to/index.xml>\n");

my $daemon = SOAP::Transport::HTTP::Daemon
  -> new (LocalPort => 8080)
  -> dispatch_to('CodeShare::Owner::(?:get|search|info|list)')
;
print "CodeShare Owner Server started at ", $daemon->url, "\n";
print "Waiting for a request...\n";
$daemon->handle;
```

Launch the daemon with the following command line:

```
C:\book>start perl cs_server.pl index.xml
```

The running program is shown in Figure 7-4.

Figure 7-4. Screenshot of the CodeShare owner server running



7.7 Implementing the CodeShare Client

The CodeShare client, like the owner server, is a Perl application that implements a simple shell for interacting with the CodeShare server and owner services. It also uses SOAP::Lite. The full source to the client is given in Appendix C. We discuss the highlights here.

First we create a proxy, shown in Example 7-40, to the CodeShare server to authenticate the user.

Example 7-40. Creating the proxy

```
my($server, $uri) =
  $ownerserver       ?       ($ownerserver                                    =>
'http://namespaces.soaplite.com/CodeShare/Owner')
                : ($codeshareserver => 'urn:Services:CodeShareServer');
my $soap = SOAP::Lite
  ->proxy($server)
  ->uri($uri);
```

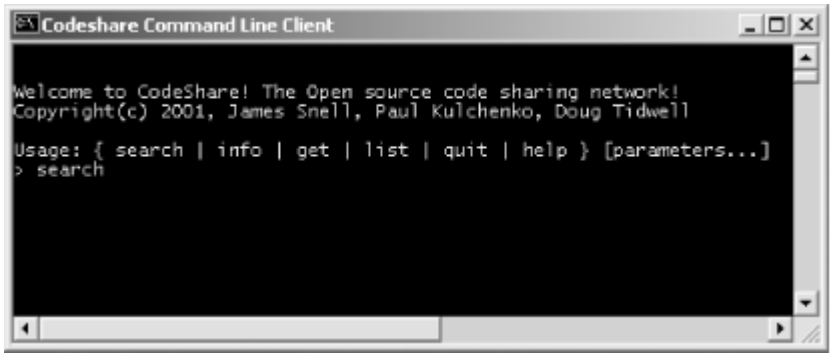
If the user logs in (doing so is completely optional), the client invokes the `login` operation exposed by the CodeShare service client interface. This returns a SAML assertion, which the client caches. We can tell whether the user is logging in based on whether he provides a username and password. Example 7-41 demonstrates this.

Example 7-41. Logging in

```
my $signature;
if ($username || $password) {
  my $response = $soap->login(
    SOAP::Data->name(credential   =>   join   ':',   $username,   $password)-
>type('base64')
  );
  die $response->faultstring if $response->fault;
  $signature = SOAP::Data->type(xml => get_signature($response));
}
```

The client is implemented as a simple shell interface, shown in Figure 7-5.

Figure 7-5. The CodeShare client shell interface



We create this shell using a `while` loop (shown in Example 7-42) to read input, work out what to do, and do it. The loop:

1. Waits for the user to enter a command (`search`, `info`, `get`, `list`, `quit`, or `help`).
2. Checks to see which command was entered.
3. Invokes the SOAP operation.
4. If the `get` command was issued, the resulting list of items is looped through and the items are returned using a simple HTTP-GET operation to the CodeShare owner server.

Example 7-42. The loop at the heart of the client

```
while (defined($_ = shift || <>)) {
  next unless /\w/;                              # must have a command
  my($method, $modifier, $parameters) =                 # split input
     m!^\s*(\w+)(?:\s*/(\w*)\s)?\s*(.*)!;

  last if $method =~ /^q(?:uit)?$/i;             # handle quit command
  help(  ), next if $method =~ /^h(?:elp)?$/i; # handle help comma


  # call the SOAP method

  my $res = eval "\$soap->$method('$parameters', '$modifier',
                  \$signature || (  ))";

  # check for errors
  $@                               and print(STDERR join "\n", $@, ''), next;
  defined($res) && $res->fault     and print(STDERR join "\n",
                                         $res->faultstring, ''), next;
  !$soap->transport->is_success and print(STDERR join "\n",
                                         $soap->transport->status, ''), next;

  # check for result
  my @result = @{$res->result} or print(STDERR "No matches\n"), next;

  foreach (@result) {
    print STDERR "$_->{type}: ",
          join(', ', $_->{Title} || (), $_->{path} || (  )), "\n";
    if ($method eq 'get') {
      if ($_->{type} eq 'directory') { File::Path::mkpath($_->{path}) }
```

```
      if ($_->{type} eq 'file') {
        open(F, '>'. $_->{path}) or warn "$_->{path}: $!\n";
        print F $_->{file};
        close(F) or warn "$_->{path}: $!\n";
      }
    } elsif ($method eq 'info') {
      foreach my $key (grep {$_ !~ /^(?:type|path)/} keys %$_) {
        print "  $key: $_->{$key}\n";
      }
    }
  }
} continue {
  print STDERR "\n> ";
}
```

7.8 Seeing It in Action

You can download an archive of the code from the O'Reilly web site (http://www.oreilly.com/catalog/progwebsoap/), and run the CodeShare service yourself. First make sure that you have properly installed Apache SOAP, Perl, and SOAP::Lite. Then run the `codeshare.bat` script for Windows or the `codeshare.sh` shell script for Unix. These scripts deploy the CodeShare services and launch the various components, allowing you to experiment with the shell client.

7.9 What's Missing from This Picture?

That was a lot of code, but even with all the programs we've written for the CodeShare server, we still haven't used all the web services components. There's no UDDI, and we haven't used the features of many P2P services, such as presence and asynchronous messaging.

7.9.1 Where Is UDDI?

One piece that is conspicuously missing is UDDI. Nowhere in the CodeShare example we've laid out so far is there any mention of how UDDI fits into the picture. This shows that none of these web services technologies are overly dependent upon one another. If UDDI is not useful in a given situation, leave it out. If SAML hadn't been useful, we would have left that out also.

In COM, CORBA, J2EE, etc., there are parts that just aren't useful, but that you still have to deal with, or at least deploy along with your sample. In web services, you are not locked into a monolithic set of technologies, but instead have a loosely coupled conglomeration of standards that make writing code easier.

UDDI could be used in this example. For instance, the CodeShare server could be listed in a public UDDI registry so that it can be more readily discovered. Additionally, since each CodeShare owner is itself a web service, owners themselves could be listed in a UDDI registry.

7.9.2 Presence and Asynchronous Messaging

Even though the CodeShare service implements a peer-to-peer architecture, there are two aspects of P2P that are missing: the concept of presence and the ability to have peers communicate with each other asynchronously.

Presence boils down to the ability of one peer to see if another peer (in this case an owner server) is currently online. The owner's HTTP daemon could very easily tell the CodeShare server when it is starting up and when it is shutting down, allowing the CodeShare server to pass that information on to users who are looking for code. This would work just like the buddy list in your Instant Messaging application.

Even if your buddies are offline, you should still be able to send messages to be delivered when they do come online. If a CodeShare user wants to get some code, and the owner of that code is currently offline, it would be nice if the owner could still get that request and handle it later. CodeShare does not yet support asynchronous communication like this.

7.10 Developing CodeShare

CodeShare has been registered as an open source project at SourceForge, a leading open source development portal. If this example has inspired you to add features to the application, you can take an active role in developing the CodeShare Service Network into a real-life infrastructure for sharing code.

To access the project at SourceForge, visit http://www.sourceforge.com/, create a user account, and search for the "CodeShare" project. To be an active contributor, you'll need to become familiar with the source code control system, called CVS. Details are available at the SourceForge web site.